Semantics of programs in a formal system with partially defined functions

Joachim Hertel

Fachbereich 10 -
Angewandte Mathematik
und Informatik -
Universität des Saarlandes

D-6600 Saarbrücken

Bericht A 78/05

März 1978

Contents
========

1. Introduction

In this report we present a formal system for an axiomatic definition of the semantics of programming languages allowing to prove correctness and termination of programs. We will consider an Algol-like language consisting of assignments, if-then-else-statements and while-loops.

Fundamental work in this area is that of Hoare [1]; the formulas in Hoare's system are of the form R{P}S, meaning that if the predicate R is true before execution of a part P of a program and if P terminates, then S is true after execution of P; note that nothing is said about termination of P in this calculus.

Our approach is something different from that of Hoare; we consider programs as functions, i.e. a program of our formal system is interpreted as a function performing a state transformation. A difficulty of this approach is that, while in usual predicate calculus all function symbols are interpreted as total functions, we need a calculus with partially defined functions. Such a calculus has been proposed by Markwald [3], besides the usual predicate symbols, this calculus contains special predicate symbols for characterizing the domain of the corresponding functions.

In applying Markwald's calculus to state transformations, we are able to express the termination of programs and we will indicate axioms allowing to prove termination.

As for the data types, we restrict ourselves in this report to integers, but a generalization for other data types does not seem to put essential difficulties. The fundamental non-logical axioms of our formal system are the axioms of arithmetic; as we will see, the axiom of induction is the most powerful tool for proving correctness and termination of programs in our system.

For those unfamiliar with Markwald's calculus, we give a summary of it in chapter 2. In chapter 3 we then present our

own system for proving correctness and termination of programs.

2. Predicate logic with partially defined functions

2.1 Some general remarks and definitions

We first introduce the logical basis for our formal system. This basis is borrowed from Markwald [3,4] and is summarized in this chapter. For more details see Markwald [3,4].

The language for Markwald's calculus consists of the following symbols:

a) the individual variables $x, y, z, x_1, y_1, z_1, \ldots$;

b) the function symbols $f_1, f_2, f_3, \ldots$;

c) the usual predicate symbols $P_1, P_2, P_3, \ldots$;

d) for each function symbol f: a special predicate symbol D_{fi}, which denote the domain of the corresponding function.

e) the logical symbols $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$, $\exists$ and $\forall$.

Next we give some recursive definitions:

1) Terms: The individual variables are terms; if $t_1, \ldots, t_k$ are terms and f_i a k-ary function symbol, then $f_i(t_1, \ldots, t_k)$ is a term.

2) Formulas: Let $t_1, \ldots, t_k$ be terms and let P_i and D_{f_i} be k-ary predicate symbols, then $P_i(t_1, \ldots, t_k)$ and $D_{f_i}(t_1, \ldots, t_n)$ are formulas;

α and β are formulas, so are $\neg\alpha$, $\alpha \wedge \beta$, $\alpha \vee \beta$, $\alpha \rightarrow \beta$, $\forall x\, \alpha$ and $\exists x\, \alpha$.

3) An interpretation I* is a mapping of the non-logical symbols of Π and refers to a non-empty set w with: [1]

$I^*(x) \in w$, for each individual variable x,

$I^*(P_i) \subseteq w^k$

$I^*(D_i) \subseteq w^k$ for each k-ary predicate symbol P_i, D_i

$I^*(f_i)$ is a mapping from $I^*(D_{f_i})$ to w

for terms t $I^*(t)$ is defined inductively, together with the relation I*bst t :[2]

I* bst x,

I*bst $f_i(t_1,\ldots,t_k)$ and $I^*(f_i(t_1,\ldots,t_k)) = I^*(f_i)$ $(I^*(t_1),\ldots,I^*(t_k))$ iff I*bst $t_1,\ldots,$I*bst t_k and $(I^*(t_1),\ldots,I^*(t_k)) \in I^*(D_{f_i})$.

i.e. in contrast to usual formal systems. $I^*(t)$ is not defined for all terms t but only for those, for which the relation I*bst t holds;

4) Next we define the relations I*erf α and I*wdl α: [3]

$$I^*\text{erf } P(t_1,\ldots,t_k) \iff I^*\text{bst } t_1,\ldots,I^*\text{bst } t_k \text{ and } (I^*(t_1),\ldots,I^*(t_k)) \in I^*(P)$$

$$I^*\text{wdl } P(t_1,\ldots,t_k) \iff I^*\text{bst } t_1,\ldots,I^*\text{bst } t_k \text{ and } (I^*(t_1),\ldots,I^*(t_k)) \notin I^*(P)$$

---

1) we use the asterisk to distinguish these interpretations I* from the interpretations in usual predicate calculus.

2) I*bst is a mnemonic abbreviation of I "bestimmt" (determines) t.

3) mnemonic abbreviations for I*"erfüllt" (satisfies) α and I*"widerlegt" (refutes) α.

$I^*erf\ \neg\alpha \iff I^*wdl\ \alpha$

$I^*wdl\ \neg\alpha \iff I^*erf\ \alpha$

$I^*erf(\alpha \wedge \beta) \iff I^*erf\ \alpha$ and $I^*erf\ \beta$

$I^*wdl(\alpha \wedge \beta) \iff I^*wdl\ \alpha$ or $I^*wdl\ \beta$

$I^*erf\ \forall\ x\ \alpha \iff I_x^{a^*}erf\ \alpha$, for all $a \in w$ and

$I^*wdl\ \forall\ x\ \alpha \iff I_x^{a^*}wdl\ \alpha$, for at least one $a \in w$,

where $I_x^{a^*}(z) = I^*(z)$, for $z \neq x$ and $I_x^{a^*}(x) = a$.

The definition of erf and wdl for the other logical symbols can easily be derived from the definition above. for example we have

$I^*erf\ (\alpha \rightarrow \beta) \iff I^*wdl\ \alpha$ or $I^*erf\ \beta$

$I^*wdl\ (\alpha \rightarrow \beta) \iff I^*erf\ \alpha$ and $I^*wdl\ \beta$.

5) A formula α is said to be universally valid, iff $I^*erf\ \alpha$ holds for all interpretations I^*.

6) $I^*bst\ \alpha \iff I^*erf\ \alpha$ or $I^*wdl\ \alpha$.

7) A formula α is said to definite, iff $I^*bst\ \alpha$ holds for all I^*.

As you see, a formula α may be universally valid, even though a subformula is not definite, for example let $\alpha \equiv \beta \rightarrow \gamma$ with some tantology γ and not definite formula β .

It follows immediately from the definitions, that the set of definite formulas is closed under logical operations.

2,2 General interpretations

Because the formulas of the calculus presented so far do not differ from the formulas of a usual formal system, they can also be interpreted in the usual way. In order to distinguish such interpretations from the (partial) inpterpretations introduced above, they may be called general or g-interpretations.
For a general interpretation I we will write I Erf α and I Wdl α instead of I erf α and I wdl α.

For each g-interpretation I there is a unique partial interpretation I^* , which is the restriction of I on terms t with I bst t.

From these two different kinds of interpretations we deduce a general and a partial notion of logical inference.

$\mathfrak{A} \vdash \alpha \iff (I\ Erf\ \mathfrak{A} \Rightarrow I\ Erf\ \alpha)$, for all g-interpretations I and

$\mathfrak{A} \vdash^* \alpha \iff (I^*erf\ \mathfrak{A} \Rightarrow I^*erf\ \alpha)$, for all (partial) interpretations I^*,

where α is a formula, $\mathfrak{A}$ a set of formulas and the relations I Erf $\mathfrak{A}$ and I^*erf $\mathfrak{A}$ are defined in the usual way.

As for the connection between these two notions of logical inference, Markwald proves the following properties

(i) $\mathfrak{A} \vdash \beta \Rightarrow \mathfrak{A} \vdash^* \beta$, if β is definite

(ii) $\mathfrak{A} \vdash^* \beta \Rightarrow \mathfrak{A} \vdash \beta$, if all $\alpha \in \mathfrak{A}$ are definite and so specially for a definite formula β:

(iii) $\vdash \beta \iff \vdash^* \beta$.

2.3 Some decidability results

Markwald has shown in [3] the following results:

1. The set of definite formulas is not decidable.
2. The set of definite formulas is recursively enumerable.
3. The set of universally valid formulas is recursively enumerable.

Let $\vdash$ be the usual notion of deduction in predicate logic. We then define a new notion of deduction:

$\mathfrak{A} \overset{*}{\vdash} \beta \iff \mathfrak{A} \vdash \beta$ and β is definite.

Because of the recursive enumerability of $\mathfrak{A} \vdash \beta$ (relative to $\mathfrak{A}$) and result (2.) of Markwald, $\mathfrak{A} \overset{*}{\vdash} \beta$ is also recursively enumerable and therefore representable by rules of inference. It is easy to show that the following holds:

a) $\mathfrak{A} \overset{*}{\vdash} \beta \Rightarrow \mathfrak{A} \overset{*}{\nvdash} \beta$ and

b) $\mathfrak{A} \overset{*}{\nvdash} \beta \Rightarrow \mathfrak{A} \overset{*}{\vdash} \beta$, if $\mathfrak{A} \cup \{\beta\}$ is a set of definite formulas.

But for the formal system presented here, there remains a difficulty; a formal system like predicate calculus has the following properties:

(i) the set of formulas is decidable,
(ii) the set of formulas is definite.
(iii) the formal system is complete, i.e. each formula which follows from $\mathfrak{A}$, is also derivable from $\mathfrak{A}$.

In this formal system , condition (ii) is not fulfilled. Therefore we redefine the calculus by adding the condition, that a formula has to be definite; unfortunately, while condition (ii) is now fulfilled, condition (i) is no longer. Actually we have to live with the somewhat weaker condition

(i') the set of (definite) formulas is recursively enumerable.

2.4 A syntactic definition of definite formulas

Up until now we have our new notion of formula - the definite formulas - defined only semantically (over the interpretations). We now give a syntactical definition of definite formulas, i.e. we will construct an algorithm to recursively enumerate the definite formulas.

First we introduce a sequence calculus for terms, the *-term calculus. It consits of two schemes of rules, the first without, the second with premises:

(1) $\emptyset * x$, for all individual variables x, where $\emptyset$ denotes the empty set

(2)
$$\frac{\begin{array}{l} D_1 * t_1 \\ \vdots \\ D_k * t_k \end{array}}{\bigcup_{j=1}^{k} D_j \cup \{D_{f_i}(t_1,\ldots,t_k)\} * f_i(t_1,\ldots,t_k)}$$

It is easy to verify, that the following holds:

If we have $D * t$, then D is definite and for each interpretation I* one has

$$I \text{ erf } D \iff I \text{ bst } t.$$

Using this *-term calculus we will now define recursively the functions B, E and W, which map the set of definite formulas in itself. The meaning of these functions is illustrated by the following properties:

(i) $I^* \text{bst } \alpha \iff I \text{ Erf } B(\alpha)$

(ii) $I^* \text{erf } \alpha \iff I \text{ Erf } E(\alpha)$

(iii) $I^* \text{wdl } \alpha \iff I \text{ Erf } W(\alpha)$

1) If $\alpha \equiv P(t_1,\ldots,t_n)$ with $D_1 * t_1,\ldots,D_n * t_n$:

$$B(\alpha) = \bigcup_{i=1}^{n} D_i$$

$$E(\alpha) = \alpha \wedge B(\alpha)$$

$$W(\alpha) = \neg\alpha \wedge B(\alpha)$$

2) If $\alpha \equiv \neg\beta$:

$$B(\alpha) = B(\beta)$$

$$E(\alpha) = W(\beta)$$

$$W(\alpha) = E(\beta)$$

3) If $\alpha \equiv \beta \wedge \gamma$:

$$B(\alpha) = (E(\beta) \wedge E(\gamma)) \vee (W(\beta) \vee W(\gamma))$$

$$E(\alpha) = E(\beta) \wedge E(\gamma)$$

$$W(\alpha) = W(\beta) \vee W(\gamma)$$

4) If $\alpha \equiv \forall x\ \beta$:

$$B(\alpha) = \forall x\ E(\beta) \vee \exists x\ W(\beta)$$

$$E(\alpha) = \forall x\ E(\beta)$$

$$W(\alpha) = \forall x\ W(\beta).$$

The properties (i), (ii) and (iii) above can be derived easily using the definitions of the relations bst, erf and wdl; from (i) it follows in particular:

α is definite $\iff$ $B(\alpha)$ is g-universally valid.

The enumerating algorithm for the definite formulas then follows from the recursive enumerability of the g-general valid formulas of usual predicate calculus.

In the next chapter we will apply the calculus presented here to develop a formal system Π for the description of the semantics of programming languages.

3. The formal system Π

3.1 Introduction

The language L_Π of Π contains the following non-logical symbols:

(i) some freely chosen function symbols for arithmetic, for example S (successor), + and *;

(ii) the function symbols p_i^j and k_i, which will be interpreted as projection functions and combinators respectively, for each i and j with $1 \leq j \leq i$;.

(iii) the function symbols denoting the programs, whose syntax will be described later;

(iv) some predicate symbols for arithmetic like $<, \leq, >$ and $\geq$;

(v) the predicate symbols T_n, $n \geq 1$, for describing n-tuples;

(vi) for each function symbol f a predicate symbol C_f for the description of repeated application of f;

(vii) for each function symbol f a predicate symbol D_f describing the domain of f.

Next we get the definition of the syntactically correct programs of Π, already mentioned under (iii):

<program> ::= begin <statement> end

<statement> ::= <assignment>|<conditional>|<while>|<sequence>

<assignment> ::= <variable> := <term>

<conditional> ::= if<test>then <statement>else<statement>fi

<while> ::= while<test>do<statement>od

<sequence> ::= <statement>;<statement>

<variable> ::= $x_0|x_1|x_2|\ldots$

For <term> may be substituted any term of Π, for <test> any formula without quantifiers.

As you will see, the effect resulting from substitution of an undefined term for <term> is easily describable in this formalism.
For abbreviation we will often use the symbols "{" and "}" instead of "begin" and "end".

In this work we restrict ourselves to a single data type, the natural numbers. But because programs are interpreted as state transformations, we have in Π still another sort of objects, the states, which may be considered as n-tuples of natural numbers.

3.2 The axioms

Next we explain the axioms of Π; note that all axioms should be definite formulas. As for the logical axioms, we take for example the axioms of Shoenfiled [5] with the exception of the equality axiom, which is replaced by

$$x = y \wedge D_f(x) \rightarrow f(x) = f(y)$$

for all function symbols and all programs f. We need this modification, because the original equality axiom is not necessarily definite.

3.2.1 Axioms for auxiliary functions and predicates

As for the non-logical axioms, there are first those characterizing the auxiliary functions and predicates viz. the "n-tuple" predicates T_i, the projection functions p_i^j, $j \leq i$, and the "combinators" k_i; intuitively $T_n(x)$ expresses that "x is a n-tuple", $p_i^j(x)$ denotes the j-th component of the i-tuple x and $k_i(x_1,\ldots,x_i)$ combines the i 1-tuples into an i-tuple.

1) $T_n(x) \leftrightarrow D_{p_n^i}(x)$, for all $i \leq n$

2) $T_1(x_1)\wedge\ldots\wedge T_1(x_n) \leftrightarrow D_{k_n}(x_1,\ldots,x_n)$, for all n.

These two axioms characterize the domains of the functions P_n^i and k_n.

3) $T_n(x) \rightarrow T_1(P_n^i(x))$, for all $i \leq n$

4) $T_1(x_1) \wedge \ldots \wedge T_1(x_n) \rightarrow T_n(k_n(x_1,\ldots,x_n))$ for all n

5) $T_1(x_1) \wedge \ldots \wedge T_1(x_n) \rightarrow P_n^i(k_n(x_1,\ldots,x_n)) = x_i$ for all $i \leq n$

6) $T_n(x) \rightarrow k_n(P_n^1(x),\ldots,P_n^n(x)) = x$, for all n.

3.2.2 Axioms for the fundamental data type

The second group of non-logical axioms describes our fundamental data type, viz. the natural numbers (see e.g. Schoenfiled [5]):

7) $T_1(x) \wedge T_1(y) \leftrightarrow D_+(x,y)$

8) $T_1(x) \wedge T_1(y) \leftrightarrow D_*(x,y)$

9) $T_1(x) \leftrightarrow D_S(x)$

These axioms characterize the domain of the functions +, * and S (successor function).

10) $T_1(o) \wedge (T_1(x) \rightarrow T_1(S(x))$

11) $T_1(x) \rightarrow \neg(S(x) = o)$

12) $T_1(x) \wedge T_1(y) \rightarrow (S(x) = S(y) \rightarrow x = y)$

13) $T_1(x) \rightarrow x + o = x$

14) $T_1(x) \wedge T_1(y) \rightarrow (x + S(y) = S(x + y)$

15) $T_1(x) \rightarrow x * o = o$

16) $T_1(x) \wedge T_1(y) \rightarrow x \cdot S(y) = (x * y) + x$

17) $T_1(x) \rightarrow \neg(x < o)$

18) $T_1(x) \wedge T_1(y) \rightarrow (x < S(y) \leftrightarrow x < y \vee x = y)$

19) $T_1(x) \wedge T_1(y) \rightarrow (x < y \vee x = y \vee y < x)$

20) $T_1(x) \rightarrow ((A_x[o] \wedge \forall x\,(A \rightarrow A_x[S(x)])) \rightarrow A)$

3.2.3 Axioms for the semantics of programs

The third group of non-logical axioms characterizes the program constructs and their termination.

21) $D_F(x) \rightarrow (T_n(x) \leftrightarrow T_n(F(x)))$, for all programs F and $n \geq 1$,

i.e. a program is a 1-ary function F mapping x (where x is interpreted a store of n memory cells) into another store F(x).

22) $T_n(x) \rightarrow D_F(x)$, for all programs F with at least n variables.

Assignment statement:

23) $T_n(x) \rightarrow (D_{\{x_{i_o} := f(x_{i_1},\ldots,x_{i_k})\}}(x) \leftrightarrow D_f(P_n^{i_1}(x),\ldots,P_n^{i_k}(x)))$

for all assignments, $1 \leq i_j \leq n$ and $k \leq n$;

Intuitively, an assignment statement terminates, if the term on the right hand side is definite.

24) $T_n(x) \rightarrow (D_{\{x_{i_o},\ldots,x_{i_k}\}}(x) \rightarrow \{x_{i_o} := f(x_{i_1},\ldots,x_{i_k})\}(x) =$

$k_n(P_n^1(x),\ldots,f(P_n^{i_1}(x),\ldots,P_n^{i_k}(x)),\ldots,P_n^n(x)))$

with $1 \leq i_j \leq n$ and $f(P_n^{i_1}(x),\ldots,P_n^{i_k}(x)$ in i_o-th position as argument of k_n; i.e. in the store x the value of x_{i_o} is substituted by the value of $f(P_n^{i_1}(x),\ldots,P_n^{i_k}(1))$.

Conditional statement:

25) $T_n(x) \rightarrow (A_{x_{i_1}\ldots x_{i_k}}[P_n^{i_1}(x),\ldots,P_n^{i_k}(x)] \rightarrow$

$(D_{\{\underline{\text{if}}\ A\ \underline{\text{then}}\ F_1\ \underline{\text{else}}\ F_2\ \underline{\text{fi}}\}(x)} \leftrightarrow D_{F_1}(x)))$.

with $1 \leq i_j \leq n$, where $A_{x_{i_1}\ldots x_{i_k}}[P_n^{i_1}(x),\ldots,P_n^{i_k}(x)]$

denotes the formula in which all occurrences of x_{i_j}, $1 \leq j \leq k$, in A are substituted by the corresponding $P_n(x)$; i.e. if A is true with store x, then {<u>if</u> A <u>then</u> F_1 <u>else</u> F_2 <u>fi</u>} (x) terminates if and only if $F_1(x)$ terminates, and then is {<u>if</u> A <u>then</u> F_1 <u>else</u> F_2 <u>fi</u>}(x) = $F_1(x)$:

26) $T_n(x) \rightarrow (A_{x_{i_1}\ldots x_{i_k}}[P_n^{i_k}(x),\ldots,P_n^{i_k}(x)] \wedge D_{F_1}(x) \rightarrow$

$\{\underline{\text{if}}\ A\ \underline{\text{then}}\ F_1\ \underline{\text{else}}\ F_2\ \underline{\text{fi}}\}(x) = F_1(x))$

and analog axioms are needed for $\neg A_{x_{i_1},\ldots,x_{i_k}}[P_n^{i_1}(x),$ $\ldots,P_n^{i_k}(x)]$

While-statement:

For the while-statement we need first three auxiliary axioms, that describe a predicate $C_F(x,y,n)$ meaning $y = F^n(x)$:

27) $C_F(x,y,o) \leftrightarrow x = y$

28) $C_F(x,y,1) \leftrightarrow D_f(x) \wedge y = F(x)$

29) $C_F(x,y,n+1) \leftrightarrow \exists z(C_F(x,z,n) \wedge C_F(z,y,1) \wedge T_1(n))$,

for all programs F.

With these predicates we can now describe the semantics of the while-statement:

30) $T_n(x) \rightarrow (D_{\{\underline{\text{while}}\ B\ \underline{\text{do}}\ F\ \underline{\text{od}}\}}(x) \leftrightarrow \exists y \exists m(C_F(x,y,m) \wedge \neg B_{x_{i_1},\ldots,x_{i_k}}[P_n^{i_1}(y),\ldots,P_n^{i_k}(y)]))$

where $x_{i_1},\ldots,x_{i_k}$ are all variables occurring in test B; i.e. a while loop {while B do F od} terminates with an argument x, if there is a natural number m, such that B is false after m executions of the loop F.

31) $T_n(x) \wedge D_{\{\underline{\text{while}}\ B\ \underline{\text{do}}\ F\ \underline{\text{od}}\}}(x) \rightarrow (\{\underline{\text{while}}\ B\ \underline{\text{do}}\ F\ \underline{\text{od}}\}(x) = y$

$\leftrightarrow \exists m(C_F(x,y,m) \wedge \forall u \forall l(l < m \wedge C_F(x,u,l) \rightarrow B_{x_{i_1},\ldots,x_{i_k}}$

$[P_n^{i_1}(n),\ldots,P_n^{i_k}(n)] \wedge \neg B_{x_{i_1},\ldots,x_{i_k}}[P_n^{i_1}(y),\ldots,P_n^{i_k}(y)]))$

where $x_{i_1},\ldots,x_{i_k}$ are all varialbes occuring in test B; that is {while B do F}(x) = y, if there is an m such that $y = F^m(x)$, B(y) is false and m is the smallest number with these properties.

Component statement:

32) $D_{\{F_1;F_2\}}(x) \leftrightarrow D_{F_1}(x) \wedge D_{F_2}(F_1(x))$

33) $D_{\{F_1;F_2\}}(x) \rightarrow \{F_1;F_2\}(x) = F_2(F_1(x))$

for all statements F_1, F_2.

The reader may already have noticed, that some of the axioms of this formal system are not definite in our strong sense, e.g. axiom 3

$$T_n(x) \rightarrow T_1(P_n^i(x))$$

because there are interpretations of the predicate symbols T_n and T_1 and the function symbol P_n^i such that P_n^i is not defined for x. On the other hand whenever axiom 1 viz.

$$T_n(x) \rightarrow D_{P_n^i}(x) ,$$

is true under an interpretation I, axiom 3 is defined for that interpretation. Hence we may avoid the difficulty mentioned, if we form a conjunction of these critical axioms. But for simplicity and clarity we have written all these axiom groups as single axioms.

With this formal system,we now want to proof as an example, correctness and termination of a simple program.

4. An example

As an example we take a program for computing the factorial:

$$G \equiv \{x_1 := 1;\ x_3 := 1;\ \underline{\text{while}}\ x_1 \leq x_2\ \text{do}\ x_3 := x_1 * x_3;\ x_1 := x_1 + 1\ \underline{\text{od}}\}$$

where x_1 is an auxiliary variable, x_2 is the argument and x_3 is the result,

First we show the termination of G, i.e.

Proposition: $\vdash T_3(x) \rightarrow D_G(x)$

Proof: $\vdash D_G(x) \leftrightarrow D_{G_1}(x) \wedge D_{G_2}(G_1(x))$, by axiom 32,

where $G_1 \equiv \{x_1 := 1;\ x_3 := 1\}$ and $G_2 \equiv \{\underline{\text{while}}\ x_1 \leq x_2\ \underline{\text{do}}\ x_3 := x_1 * x_3;\ x_1 := x_1 + 1\ \underline{\text{od}}\}$

To show $\vdash T_3(x) \rightarrow D_{G_1}(x)$ is trivial; to show the rest, we first compute $G_1(x)$:

$$\vdash T_3(x) \rightarrow \{x_1 := 1;\ x_3 := 1\}(x) = \{x_3 := 1\}(k_3(1, P_3^2(x), P_3^3(x)))$$

by axioms 24 and 33

$\vdash T_3(x) \rightarrow G_1(x) = k_3\ (1,P_3^2(x),1)$ by axiom 24

It reamins to be shown that $G_2(G_1(x))$ terminates, i.e.

$\vdash T_3(x) \rightarrow \exists z \exists n(C_F(k_3(1,P_3^2(x),1),z,n) \wedge \neg P_3^1(z) \leq P_3^2(z))$

with $F = \{x_3 := x_1 \cdot x_3;\ x_1 := x_1 + 1\}$

We show the following

Lemma: $\vdash T_3(x) \rightarrow C_F(k_3(1,P_3^2(x),1),k_3(P_3^2(x)+1,P_3^2(x),$
$fac(P_3^2(x)),P_3^2(x)))$

Proof: We prove the more general proposition

$\vdash T_3(x) \rightarrow C_F(k_3(1,z,1),k_3(P_3^2(x)+1,z,fac(P_3^2(x))),P_3^2(x))$

by induction over $P_3^2(x)$

(i) $P_3^2(x) = o$:

$\vdash T_3(x) \rightarrow C_F(k_3(1,z,1),k_3(1,z,1),o)$

this follows immediately from axiom 27.

(ii) Let $n = P_3^2(x)$ and suppose

$\vdash T_3(x) \rightarrow C_F(k_3(1,z,1),k_3(n+1,z,fac(n)),n)$

We then have to show

$\vdash T_3(x) \rightarrow C_F(k_3(1,z,1),k_3(n+2,z,fac(n+1)),n+1)$

From the definition of F we have

$F(k_3(n+1,z,fac(n))) = k_3(n+z,z,fac(n+1))$ and then by axiom 28

$\vdash T_3(x) \rightarrow C_F(k_3(n+1,z,fac(n)),k_3(n+z,z,fac(n+2)),1)$

and the result follows immediately from axiom 29.

So we still have to show

$$\vdash \neg P_3^1(z) \leq P_3^2(z) \qquad \text{for } z = k_3(P_3^2(x)+1, P_3^2(x), fac(P_3^2(x))).$$

But this formula follows immediately from axiom 5, and the termination of our factorial program is proved.

Next we show the correctness of G; therefore we have to show

$$\vdash T_3(x) \rightarrow P_3^3(G(x)) = fac(P_3^2(x)).$$

By the proof of termination and axiom 31 we have

$$\vdash T_3(x) \rightarrow [G(x) = k_3(P_3^2(x)+1, p_3^2(x), fac(P_3^2(x))) \leftrightarrow \exists k$$

$$(C_F(k_3(1, P_3^2(x), 1), k_3(P_3^2(x)+1, P_3^2(x), fac(P_3^2(x))), k)$$

$$\wedge ((m < k \wedge C_F(k_3(1, P_3^2(x), 1), u, m)) \rightarrow P_3^1(u) \leq P_3^2(u)))]$$

Choosing $k = P_3^2(x)$, we have to show

$$\vdash T_3(x) \rightarrow (m < P_3^2(x) \wedge C_F(k_3(1, P_3^2(x), 1), u, m) \rightarrow P_3^1(u) \leq P_3^2(u)) .$$

In the proof of termination we have already shown

$$\vdash C_F(k_3(1, z, 1), k_3(m+1, z, fac(m)), m).$$

By uniqeness of the second component of C_F, which can easily be shown by induction over m, follows

$$\vdash T_3(x) \rightarrow ((m < P_3^2(x) \wedge C_F(k_3(1, P_3^2(x), 1), u, m)) \rightarrow$$

$$u = k_3(m+1, P_3^2(x), fac(m)))$$

and

$$\vdash T_3(x) \rightarrow ((m < P_3^2(x) \wedge C_F(k_3(1, P_3^2(x), 1), u, m)) \rightarrow P_3^1(u) \leq P_3^2(u),$$

and so we have our result

$$\vdash T_3(x) \rightarrow G(x) = k_3(P_3^2(x) + 1,\ P_3^2(x),\ fac(P_3^2(x)))$$

$$\vdash T_3(x) \rightarrow P_3^3(G(x)) = fac(P_3^2(x))$$

which completes our proof of correctness.

Conclusion

We have embedded ALGOL-like programs in a formal system with partially defined functions, where the semantics of programs is described as the semantics of the corresponding function symbols, the state transformation. In our formalism we can simulate several other approaches to the semantics of programs, for example the formulas of Hoare's system, $R\{P\}S$, are translated in Π as $R(x) \wedge D_P(x) \rightarrow S(P(x))$, or the formulas of Manny/Pnueli's system of total correctness of programs [2], $\langle R(x)|P|Q(x,P(x))\rangle$, can be translated as $R(x) \rightarrow D_P(x) \wedge Q(x,P(x))$.

There is still another point of view of the formal system presented here. As in chapter 2.2 we could interprete the formulas as those of usual predicate formulas, i.e. all function symbols are interpreted as total functions, and we are interested only in those arguments x of a function symbol f, for which $D_F(x)$ can be derived. The disadvantage of this point of view is, that there are now interpretations I* of program F, such that I*(F(x)) has a defined value, even though F does not hold with argument x; i.e. we would have a lot of undesired models for our calculus.

In a later paper, we will show, that all formulas that can be proved in Hoare's system or in Manna/Pnueli's system, can also be proved in Π; we moreover will extend the system for the description of goto-statements. As we will see, this new system is then a formalization of a version of the intermittend

assertion method (see Burstall [6] and Manna/Waldinger [7]) using predicate calculus.

Acknowledgement: The author wishes to thank J. Loeckx for several discussions on the subject.

References
==========

[1] C.A.R. Hoare: An axiomatic basis for computer programming, CACM 12, No 10, 1969

[2] Z.Manna/A.Pnueli: Axiomatic approach to total correctness of programs, Acta Informatica 3, 243-263 (1974)

[3] W. Markwald; Prädikatenlogik mit partiell definierten Funktionen. Archiv für math. Logik 14 (1971), 10-23

[4] W. Markwald; Prädikatenlogik mit partiell definierten Funktionen II, Archiv für math. Logik 16 (1974), 15-22

[5] J. Shoenfield: Mathematical logic, Addison-Wesley, 1967

[6] R.M. Burstall; Programm proving as hand simulation with a little induction ,Information Processing 74, North Holland Publ. Co., pp. 3o8-312

[7] Z. Manna/R. Waldinger: In 'sometime' better than 'always'-intermittend assertions in proving programm correctness, CACM 21, no 2, 1978